Pandas

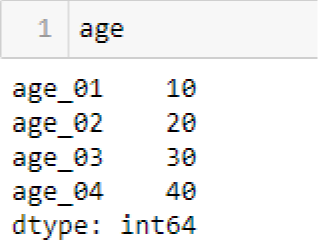
pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. The name is derived from the term "panel data"

1. Importing pandas and setting alias
    - import pandas as pd

Series

Pandas provide two convenient data structures for storing and manipulating data--Series and DataFrame. A Series is similar to a one-dimensional array whereas a DataFrame is more similar to representing a matrix or a spreadsheet table.

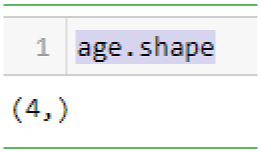
1. Lets create a pandas series consisting of index value
    - age = pd.Series([10, 20, 30, 40], index = ['age_01', 'age_02', 'age_03', 'age_04'])



```
age
```

```
age_01    10
age_02    20
age_03    30
age_04    40
dtype: int64
```

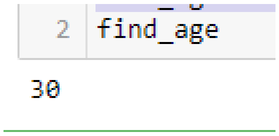
2. Checking the shape of the array
    - age.shape



```
age.shape
```

```
(4,)
```

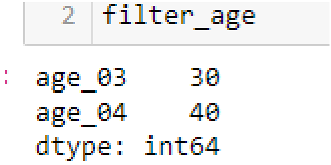
3. Accessing the value with index name
    - find_age = age.age_03



```
find_age
```

```
30
```

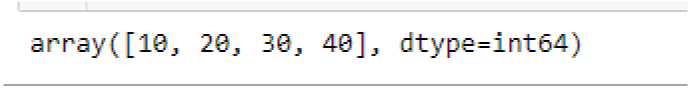
4. Filtering and getting certain value less than 20
    - filter_age = age[age > 20]



```
filter_age
```

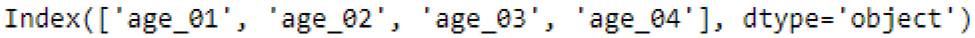
```
age_03    30
age_04    40
dtype: int64
```

5. Calling all the values of the series
    - age.values



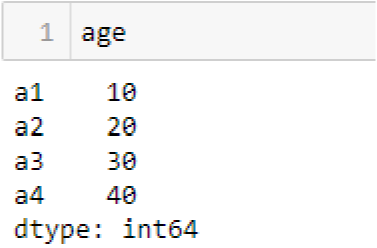
```
array([10, 20, 30, 40], dtype=int64)
```

6. Getting all the index values
    - age.index



```
Index(['age_01', 'age_02', 'age_03', 'age_04'], dtype='object')
```

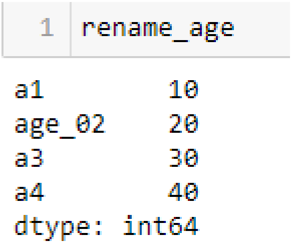
7. Lets change the index of the age series
    - age.index = ['a1', 'a2', 'a3', 'a4']



```
age
```

```
a1    10
a2    20
a3    30
a4    40
dtype: int64
```

8. Renaming the index ‘a2’ to ‘age_02’
    - rename_age = age.rename(index = {'a2':'age_02'})



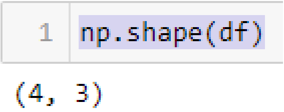
```
rename_age
```

```
a1        10
age_02    20
a3        30
a4        40
dtype: int64
```

DataFrame

Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns

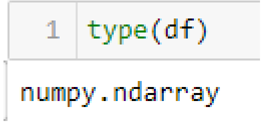
1. Importing numpy and setting alias for np
    - import numpy as np
2. Creating a simple numpy array
    - df = np.array([[20, 10, 8], [25, 8, 10], [27, 5, 3], [30, 9, 7]])
3. Lets check the shape of that array
    - np.shape(df)



```
np.shape(df)
```

```
(4, 3)
```

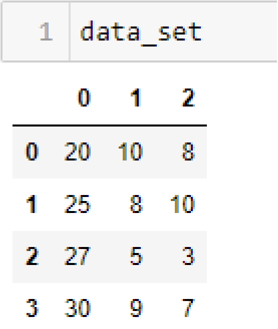
4. Checking the type of that array
    - type(df)



```
type(df)
```

```
numpy.ndarray
```

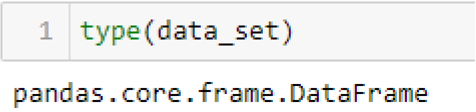
5. Now lets convert that array into a dataframe
    - data_set = pd.DataFrame(df)
6. After we convert that array into data frame the output looks like this



```
data_set
```

	0	1	2
0	20	10	8
1	25	8	10
2	27	5	3
3	30	9	7

7. Checking the type
    - type(data_set)

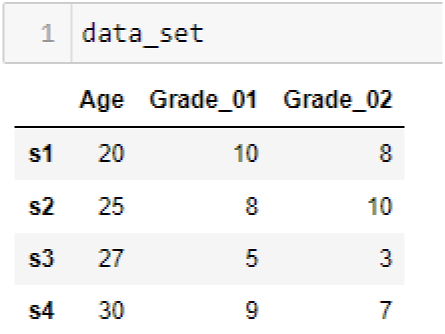


```
type(data_set)
```

```
pandas.core.frame.DataFrame
```

8. Now lets add index with column name in that particular dataset

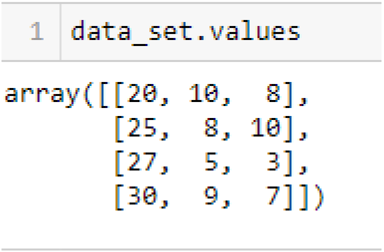
- data_set = pd.DataFrame(df, index = ['s1', 's2', 's3', 's4'], columns = ['Age', 'Grade_01', 'Grade_02'])



```
data_set
```

	Age	Grade_01	Grade_02
s1	20	10	8
s2	25	8	10
s3	27	5	3
s4	30	9	7

9. Printing the values of that dataset
   - data_set.values

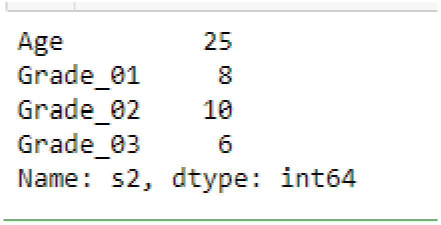


```
data_set.values
```

```
array([[20, 10,  8],
       [25,  8, 10],
       [27,  5,  3],
       [30,  9,  7]])
```

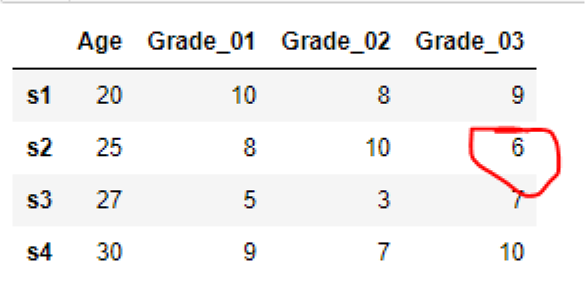
Loc and iloc

1. Getting all the values in row with index name s2
   - Data_set.loc['s2']



```
Age         25
Grade_01     8
Grade_02    10
Grade_03     6
Name: s2, dtype: int64
```

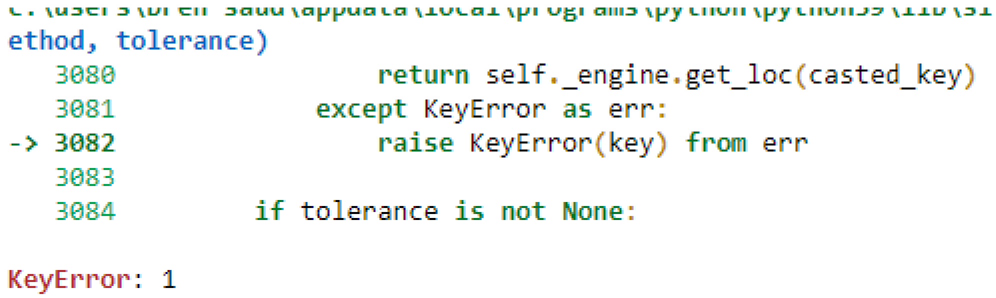
2. Selecting specific value with loc
   - data_set.loc['s2']['Grade_03']



	Age	Grade_01	Grade_02	Grade_03
s1	20	10	8	9
s2	25	8	10	6
s3	27	5	3	7
s4	30	9	7	10

The result will be “6”

3. Now we want to select the same item with index name, will we get the same result.
    - Here we know that the value 6 is in " 2nd index " .i.e. **[1]** and in ' 4th column' **[3]** index
    - data_set.loc[1][3]
      In the result we will get some error like this

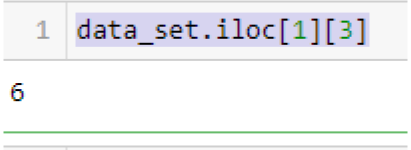


```
c:\users\bren saad\appdata\local\programs\python\python39\lib\si
ethod, tolerance)
   3080                        return self._engine.get_loc(casted_key)
   3081                    except KeyError as err:
-> 3082                        raise KeyError(key) from err
   3083
   3084                if tolerance is not None:

KeyError: 1
```

Note: loc is label-based, which means that you have to specify rows and columns based on their row and column labels. This is where iloc comes in. iloc is integer position based , we have to specify rows and columns value with integer position

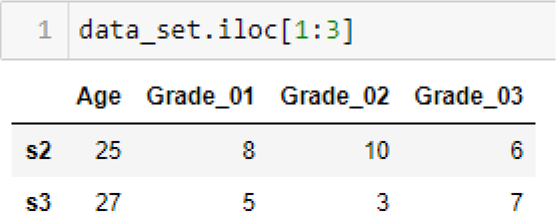
4. Achieving the same result with index values
    - data_set.iloc[1][3]



```
data_set.iloc[1][3]
```

```
6
```

5. In the same braces
    - data_set.iloc[1,3]
6. Getting values of 2nd row and all column
    - dataset.iloc[1:3]

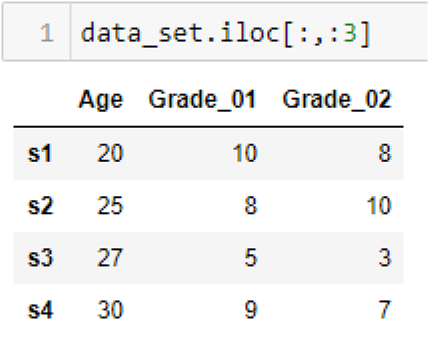


```
data_set.iloc[1:3]
```

	Age	Grade_01	Grade_02	Grade_03
s2	25	8	10	6
s3	27	5	3	7

7. Getting values of all column and upto specific rows

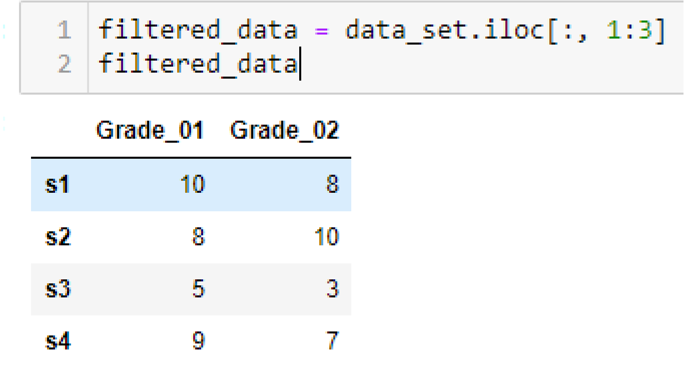
- data_set.iloc[:,:3]



```
data_set.iloc[:,:3]
```

	Age	Grade_01	Grade_02
s1	20	10	8
s2	25	8	10
s3	27	5	3
s4	30	9	7

8. Getting some filtered values
    - filtered_data = data_set.iloc[:, 1:3]

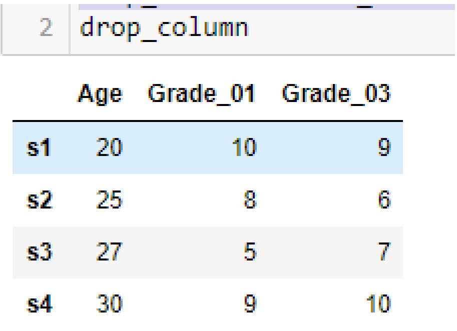


```
filtered_data = data_set.iloc[:, 1:3]
filtered_data
```

	Grade_01	Grade_02
s1	10	8
s2	8	10
s3	5	3
s4	9	7

Dropping values in dataframe

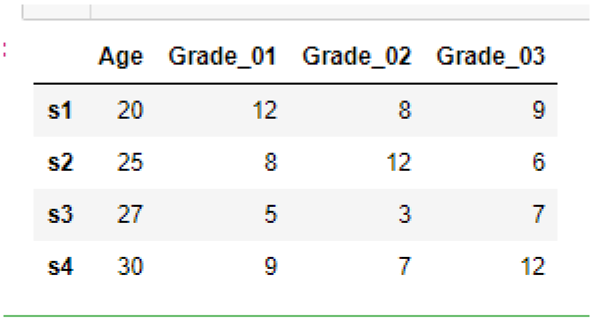
1. Dropping some column by defining its axis. As the axis for the column is 1 we give the axis as 1. If it was a row we would have given the axis as 0.
    - drop_column = data_set.drop('Grade_02', axis = 1)



```
drop_column
```

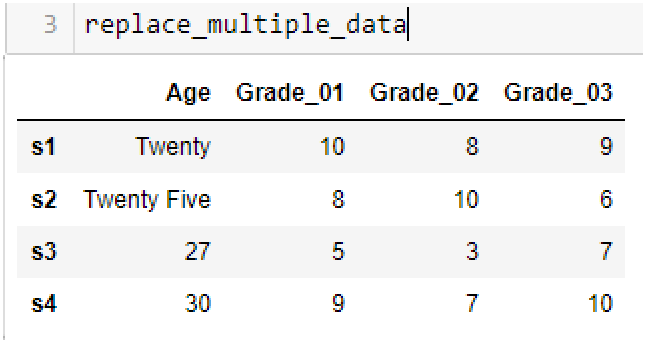
	Age	Grade_01	Grade_03
s1	20	10	9
s2	25	8	6
s3	27	5	7
s4	30	9	10

2. Replacing the values were there is 10
    - replace_data = data_set.replace(10,12)



	Age	Grade_01	Grade_02	Grade_03
s1	20	12	8	9
s2	25	8	12	6
s3	27	5	3	7
s4	30	9	7	12

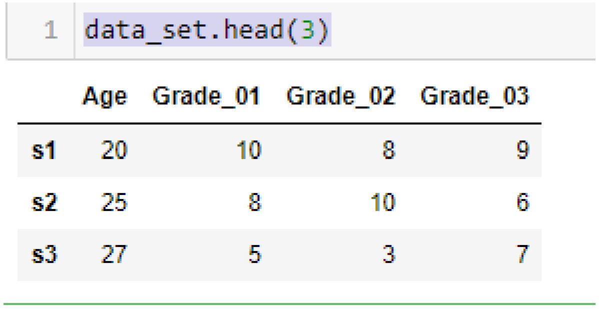
3. Replacing the data by specifying in dictionary.
    - replace_multiple_data = data_set.replace({20:'Twenty', 25:'Twenty Five'})



```
replace_multiple_data
```

	Age	Grade_01	Grade_02	Grade_03
s1	Twenty	10	8	9
s2	Twenty Five	8	10	6
s3	27	5	3	7
s4	30	9	7	10

4. Getting the top 3 values from the dataset.
    - data_set.head(3)

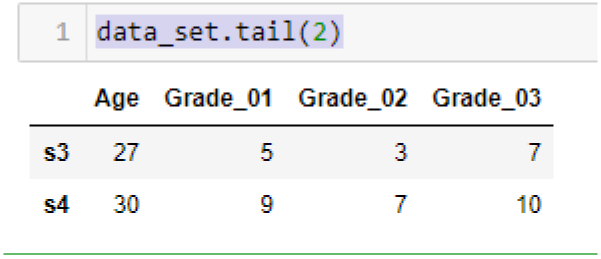


```
data_set.head(3)
```

	Age	Grade_01	Grade_02	Grade_03
s1	20	10	8	9
s2	25	8	10	6
s3	27	5	3	7

Note: by default it takes 5 and only shows five

5. Getting last two values from the bottom of the dataframe
    - data_set.tail(2)

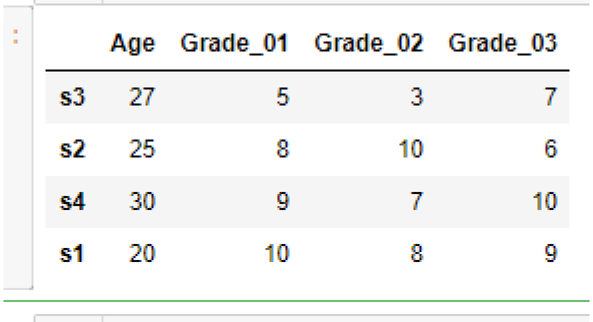


```
data_set.tail(2)
```

	Age	Grade_01	Grade_02	Grade_03
s3	27	5	3	7
s4	30	9	7	10

6. Sorting values with respect to certain column in ascending order

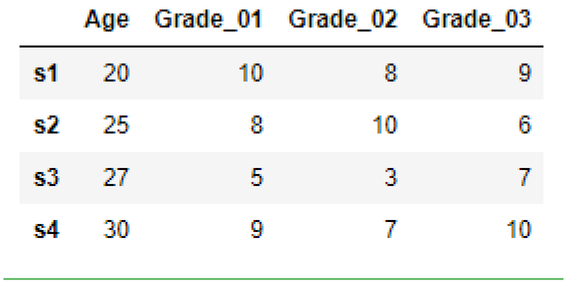
- data_set.sort_values('Grade_01',ascending = True)



	Age	Grade_01	Grade_02	Grade_03
s3	27	5	3	7
s2	25	8	10	6
s4	30	9	7	10
s1	20	10	8	9

7. Default sorting

- data_set.sort_index(axis = 0, ascending = True)



	Age	Grade_01	Grade_02	Grade_03
s1	20	10	8	9
s2	25	8	10	6
s3	27	5	3	7
s4	30	9	7	10